Knowledge Graph Enhanced Web API Recommendation via Neighbor Information Propagation for Multi-service Application Development

Zhen Chen[1,3(✉)], Yujie Li[1], Yuying Wang[1], Xiaowei Liu[1], Yifan Xing[1], Linlin Liu[2], Dianlong You[1,3], and Limin Shen[1,3]

[1] College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2] National Science Libraries, Chinese Academy of Sciences, Beijing 100864, China
[3] The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China
zhenchen@ysu.edu.cn

Abstract. In cloud era, Web APIs have been the best carrier for service delivery, capability replication and data output in multi-service application development. Currently, the number of Web APIs on the Internet is huge and growing exponentially. To enable accurate and fast Web API selection for developers, researchers have proposed a variety of Web API recommendation methods. However, existing methods cannot solve the inherent data sparsity problem well. In addition, existing methods use context information indirectly by finding neighbors or discretely through embedding techniques, while rich semantic information in the Web API ecosystem is ignored. To solve the above problems, we firstly crawl and analyze Web API data to construct a Web API knowledge graph, which laid a data foundation for alleviating the data sparsity problem. Then, we propose a knowledge graph-enhanced Web API recommendation model, so as to improve recommendation accuracy by capturing high-order structural information and semantic information. Typically, multivariate representations of user and Web API are made by the neighbor information propagation in Web API knowledge graph. The proposed model supports end-to-end learning for beneficial feature extraction. Finally, experiments results demonstrate the proposed model outperforms baselines significantly, thereby promoting the development of Web API economy.

Keywords: Multi-service application development · Web API recommendation · Knowledge graph · Neighbor information propagation

Supported by National Natural Science Foundation of China, No. 62102348, 62276226. Natural Science Foundation of Hebei Province, China, No. F2022203012, F2021203038. Science and Technology Research Project of Hebei University, No. QN2020183. Innovation Capability Improvement Plan Project of Hebei Province, No. 22567626H.

H. Gao et al. (Eds.): CollaborateCom 2022, LNICST 460, pp. 20–40, 2022.
https://doi.org/10.1007/978-3-031-24383-7_2

1 Introduction

Service-oriented architecture (SOA) is an enterprise-wide approach to agile application development as well as a collection of design principles that support loose coupling and reusability of different components in distributed applications [1]. Nowadays, SOA is changing the way of software development, which enables software development through the cloud by seamlessly composing readily interoperable service components [2], with each service component being responsible for a small subset of the application's functionality. On one hand, developers are able to invoke existing services for many required functionalities rather than write one start-to-finish set of codes after another, avoiding repeated wheel reinventing, long development cycles, and improving application elasticity. On the other hand, enterprises release their core business capabilities and valuable data in the form of services, realizing the maximum utilization of their legacy assets. Therefore, with the rise of SOA, software development has currently started undergoing a major shift from monolithic applications, to graphs of hundreds of loosely-coupled service components, and multi-service applications have become the de-facto standard for delivering today's enterprise IT applications [3].

Currently, Web API is the mainstream technology to realize SOA. Different from the traditional SOAP based Web services, Web API is based on a simpler REST architecture and has attracted great attention due to the advantages of easy access, easy composition and lightweight [4]. Web APIs serve as the fundamental building blocks of modern software that provide direct and indirect cloud services and data to developers. Hence, developers can combine different kinds of existing Web APIs to achieve required functions in almost any distributed multi-service application today [5], which greatly reduces the burden of developers and improves developer agility and productivity. For instance, developers only need to focus on their key business process of the check-in system, rather than designing and training a complex face recognition algorithm for the implementation of identity authentication module. Hence, with the increasing complexity of distributed application system, learning and invoking Web API has become a common activity and a key challenge in modern multi-service application development.

In recent years, multi-service application drives the formation and the increasing prosperity of Web API economy. As a result of such a driving force, the number of Web APIs on the Web is huge and growing exponentially [6]. Many Web API sharing platforms appear on the Internet, such as ProgrammableWeb, RapidAPI and APIList. Although rich Web API resources provide users with more choices, it is difficult for developers lacking expertise to quickly choose appropriate Web APIs that meet their personalized needs, due to the fact that there are a large number of Web APIs with similar functionalities on the Web. We note that ProgrammableWeb and other platforms provide the way of keyword and tag search, but the returned results through keyword or tag search are not satisfactory. This passive retrieval method is not conducive to meeting the needs of developers for Web APIs, thereby hindering the efficient software

development. Therefore, the active and personalized Web API recommendation has become an urgent problem to be addressed.

Among some methods that have been proposed to solve the personalized recommendation of Web API, neighborhood-based recommendation method has been widely understood and applied in the field of recommendation. Neighborhood-based recommendation is one line of collaborative filtering method [7], which works by searching a group neighbors for the target user or Web API based on the calculated similarity using the historical interaction data between user and Web API, user's preference to the target Web API can be predicted by these selected neighbors, and the final Web API recommendation list is generated based on the predicted preferences [8]. Noting that the number of available Web APIs on the Internet is huge, a single user only used a few of them, and thus the user-Web API interaction matrix is usually sparse in real Web API ecosystem. In such data sparse situations, the similarity between users in neighborhood-based methods may be exaggerated. Faced with the unavoidable problem of data sparsity, neighborhood-based method has the disadvantages of low accuracy. Model-based methods use some machine learning algorithms to train the vector of items, and then build a model to predict the probability that a user will have a demand for a Web API [9]. Although data sparsity is alleviated by avoiding similarity calculation, the interaction between users and Web APIs in real scenarios is complex and nonlinear, the simple linear dot product calculation in model-based recommendation methods limits the prediction accuracy [10]. In recent years, with the strong representational learning ability of deep learning, deep learning is able to establish the complex interaction relationship between user and Web API, which provides promising opportunities to advance Web API recommendation.

However, there are still some challenges on the research road of realizing accurate Web API recommendation. Specifically, there are the following challenges:

(1) *The widely used embedding technology in deep learning method only produces single user and API representation, which makes the inevitable data sparsity still not resolved effectively.* Existing deep learning based Web API recommendation methods usually use embedding technology to transform users, Web APIs and their attribute features into dense vectors. The interactive relationship between users and Web APIs attempts to be captured by the complex and deep network structure. However, it is easy to produce over fitting problem when the deep learning model is complex with small amount of training data, which makes the sparsity problem still unresolved and even more serious.

(2) *High-order structural information and semantic information are ignored in many existing Web API recommendation methods, which make the accuracy of Web API recommendation results still unsatisfactory.* There is a lot of contextual information in the interaction between users and APIs, but they are not fully exploited. However, in practical scenario, user, API, and contextual information are not independent of each other, but have rich

semantic and associated relationships. Using these relationships can effectively enrich and enhance the intrinsic feature representation of the user and Web API, thus providing a promising solution to solve the sparsity problem and improve accuracy.

(3) *Existing feature representation methods strongly rely on the use of feature engineering for beneficial feature extraction, which cannot achieve end-to-end training and reduce the scalability of the model.* Many existing studies believe that contextual information between user and API interaction can effectively solve the sparsity problem, but not all contextual information is beneficial. Therefore, identifying beneficial contexts is the key to improve the accuracy of recommendation. Existing methods mostly depend on the tedious manual features engineering for beneficial feature extraction. However, this method cannot be generalized to the patterns in untrained samples and cannot realize end-to-end training, and thus reducing the scalability of recommendation model.

To further improve the accuracy of recommendation, we consider introducing knowledge graph into Web API recommendation. Knowledge graph is a semantic network graph that describes various entities and their relationships in the objective world, which can be used as an effective tool to describe the entities and their relationships in Web API ecosystem. With structured Web API knowledge, knowledge graph provides the ability to analyze problem from a "relation" perspective, and provides a new way to solve the above challenges. In this paper, we introduce a knowledge graph enhanced Web API recommendation approach, named KGWARec, which makes full use of the multivariate representation of user and Web API to mimic a decision-making process where the multivariate representation is determined by the high-order structural and semantic information through neighbor information propagation in our established Web API knowledge graph. Specifically, a user/Web API as a node in knowledge graph is represented by his/her multi-hop neighbors in our constructed Web API knowledge graph through information propagation, so as to realize the multivariate representation of user and Web API. Through estimating the preference of user on the target Web API, we can generate effective Web API recommendation list for multi-service application developers. Since our embedding representation of user and Web API is a multivariate representation, which can automatically discover users' hierarchical potential interests and capture Web APIs' potential multivariate representation by iteratively propagating information in a knowledge graph, generating accurate Web API recombination results.

In conclusion, the contributions of this paper are as follows:

(1) We construct a Web API knowledge graph based on the crawled data from real-world Web API platform ProgrammableWeb, which provides a new perspective of mining potential interest of users and multivariate representation of Web APIs from the perspective of semantic relations.
(2) We develop a multivariate representation method for users and Web APIs based on knowledge graph, wherein high-order structural information and

semantic information are captured, which has a better capability to predict accurate preference of user on the target Web API in the data sparse case.
(3) We propose a knowledge graph enhanced Web API recommendation model via neighbor information propagation with an end-to-end training manner, wherein no prior knowledge of feature representation is needed in the training.

The reminder of this paper is organized as follows. We review works related to our approach in Sect. 2. We describe the Web API knowledge graph construction and formulate our recommendation task in Sect. 3. In Sect. 4, we introduce the details of the proposed approach. We through extensive experiments prove the effectiveness and superiority of our approach in Sect. 5. Finally, we conclude this paper and discuss our future works.

2 Related Works

2.1 Collaborative Filtering Based Web API Recommendation

Collaborative filtering based Web API recommendation method is a traditional recommendation method that appeared earlier and has a wider range of uses. Because the idea of collaborative filtering method is easy to understand and accept, many researchers improve the accuracy of Web API recommendation by improving collaborative filtering method. Tang et al. [11] integrate the spatial information of users and Web services in the traditional similarity calculation process. Chen et al. [12] use the user's credit and spatial context information to improve the accuracy of similarity calculation. Collaborative filtering based Web API recommendation method still mainly relies on the user's historical usage records. However, it can be noticed from observation that among the platforms that provide Web API, the number of Web APIs is very large, but the number of Web APIs used by users is small, and the user-Web API historical interaction matrix is very sparse. Satisfactory recommendation results cannot be obtained under such conditions.

2.2 Model Based Web API Recommendation

Different from neighborhood based methods that only use neighbors for prediction, model based methods usually use the overall data to learn a prediction model to solve the high sparsity problem. The most representative matrix factorization (MF) method projects the high-dimensional sparse matrix into two low-dimensional dense matrices, and uses the obtained matrices to discover the user's preference for Web API [13]. Fletcher [14] propose a MF method that considers the quality features of Web APIs, which enhances the quality and the diversity of Web API recommendations. Chen et al. [15] assume that similar neighbors have similar latent feature, and a MF model that considers the neighborhood is proposed by integrating the diversified neighbors of the API.

Although model-based methods can effectively alleviate the problem of data sparsity and improve the accuracy of recommendations by introducing similar relationships and contextual information, model-based Web API recommendation methods still have the problem that the performance decreases as the sparsity and dimensionality of the input data increase. And it is also difficult to accurately determine the optimal dimension of the low-dimensional dense matrices in practical applications. In addition, Web API recommendation systems often lack explicit scoring data and usually need to use implicit interaction data. However, researchers have found that MF is more suitable for explicit feedback matrices, while the results of the feedback matrix for implicit behavior are not ideal [16]. Therefore, model-based methods still have great limitations in Web API recommendation.

2.3 Deep Learning Based Web API Recommendation

Deep learning method integrate heterogeneous side information such as feedback data, social networks, attributes and so on to optimize the recommendation results. In recent years, there have been many applications of deep learning in Web API recommendation: Xiong et al. [17] propose a deep hybrid service recommendation (DHSR) model that integrates MLP and text similarity calculations to learn the nonlinear relationship between mashups and Web APIs. Cao et al. [18] use Doc2Vec and attention factorization machine (AFM)to extract and identify the importance of each feature of Web API and then realize the modeling of multi-dimensional information. Zhao et al. [19] propose a Web API recommendation method using feature integration and learning ranking. This method uses the text, nearest neighbor, Web API specific feature and tag of Web APIs to estimate the correlation between Web APIs. Web API recommendation based on deep learning converts information such as attributes and context into a single feature vector input model, and there are still a large number of potential associated semantic information between various information that does not play a role. Knowledge graph can show the complete ecology of a field in the real world, mainly through the relationship to complete the connection between things. Therefore, consider introducing knowledge graphs to further improve the performance of Web API recommendation methods.

2.4 Knowledge Graph Based Web API Recommendation

Recently, researchers began to pay attention to how to make full and in-depth use of the information in the knowledge graph for knowledge perception recommendation [20]. Due to the existence of a large number of potential users, association information and attribute information between various individuals in Web API ecosystem, the recommendation method based on knowledge graph has also begun to emerge in the field of Web API recommendation. Kwapong et al. [21] presented a knowledge graph based framework for Web API recommendation, and show how to use knowledge graph to improve Web API recommendation with side information. Based on the above framework, a method for mashup tag

recommendation is further proposed [22]. Wang et al. [23] designed a knowledge graph schema to encode the mashup-specific contexts and model the mashup requirement with graphic entities, and then exploit random walks with restart to assess the potential relevance between the mashup requirement and the Web APIs according to the knowledge graph. Geng [24] used KGCN to mine the high-order relationship between Web services and mashup preference, and adopt Doc2Vec model to get the semantics of mashups. Inspired by the above work, this paper uses the knowledge graph neighbor information propagation method to obtain the neighbors of the Web API and the user, and makes full use of the attribute information and colorful relationship information of various entities in the Web API knowledge graph to generate multiple representations of the user and the Web API, and then achieve accurate recommendation.

3 Preliminaries

3.1 Web API Knowledge Graph Construction

Knowledge graph provides the ability to make Web API recommendation from a "relation" perspective. We construct a Web API knowledge graph according to the following steps: data acquisition, knowledge extraction and knowledge integration.

Stage 1: *Data acquisition.* We collect data from ProgrammableWeb, which contains name, category, label and other detailed information of Web APIs, mashups, users and so on.

Stage 2: *Knowledge extraction.* We propose a Web API knowledge extraction method based on top-down analysis. Specifically, we first use graph mapping to define the top-level entity relationship pattern, and then extract the entity and relation knowledge according to the relationship pattern. Finally, seven types of entities and eight types relations are extracted.

Stage 3: *Knowledge integration.* Based on the crawled data and the extracted entity and relation knowledge, we form the knowledge of Web API ecology in the form of triple (head entity-relationship-tail entity).

Based on the above process, the statistics of the Web API knowledge graph we built are shown in Table 1.

3.2 Problem Formulation

For the Web API ecosystem in real world, we have a set of users $U = \{u_1, u_2, ...\}$, a set of Web APIs $A = \{a_1, a_2, ...\}$ and the user-Web API interaction matrix $Y = \{y_{ua} \mid u \in U, a \in A\}$. The user-Web API interaction matrix Y is defined by whether a user has used a Web API. The multi-service application Mashup consists of Web APIs with various functions, which builds usage relationship between Web APIs invoked by the Mashup and the Mashup developer. Specifically, if user u has used the Web API a, then $y_{ua} = 1$, otherwise, $y_{ua} = 0$. Web API knowledge graph is a semantic network and can be expressed as a collection

Table 1. Statistics of the built Web API knowledge graph

Name	Knowledge type	Count
Web API	Entity	21,126
User	Entity	147,566
Web API Category	Entity	421
Web API Secondary Category	Entity	925
Web API Tag	Entity	7,617
Mashup	Entity	7,621
Provider	Entity	6,529
(Web API, belong, Category)	Triple relation	21,070
(Web API, belong, SCategory)	Triple relation	49,120
(Web API, has, Tag)	Triple relation	50,878
(User, follow, Web API)	Triple relation	567,044
(Provider, provide, Web API)	Triple relation	26,048
(User, follow, Mashup)	Triple relation	15,180
(User, develop, Mashup)	Triple relation	5,516
(Mashup, invoke, Web API)	Triple relation	13,138

of triples $G\{(h, r, t)\}$, where (h, r, t) denotes the knowledge triple, $h \in \epsilon$, $r \in R$, $t \in \epsilon$ are the head entity, relation and tail entity of a knowledge triple respectively, and the symbol ϵ and R represent the entity set and relation set in the Web API knowledge graph respectively.

The knowledge graph enhanced Web API recommendation problem can be formulated as: given user-Web API interaction matrix Y and Web API knowledge graph G, our goal is to predict whether user u has potential interest to the Web API a that user u has not used before. Specifically, our task is to learn a prediction model

$$\hat{y}_{u,a} = \mathcal{F}(u, a \,|Y, \mathcal{G}\,) \tag{1}$$

where $\hat{y}_{u,a}$ denotes the preference probability that user u may use Web API a in the future.

4 Methodology

In this section, we describe the proposed KGWARec in detail. We first introduce the overall framework of KGWARec. Then, we present the modeling process of KGWARec from three aspects: preprocessing layer, multivariate representation layer and prediction layer. Finally, the learning algorithm of KGWARec is described.

4.1 Framework

Figure 1 shows the framework of KGWARec, consisting of preprocessing layer, multivariate representation layer and prediction layer.

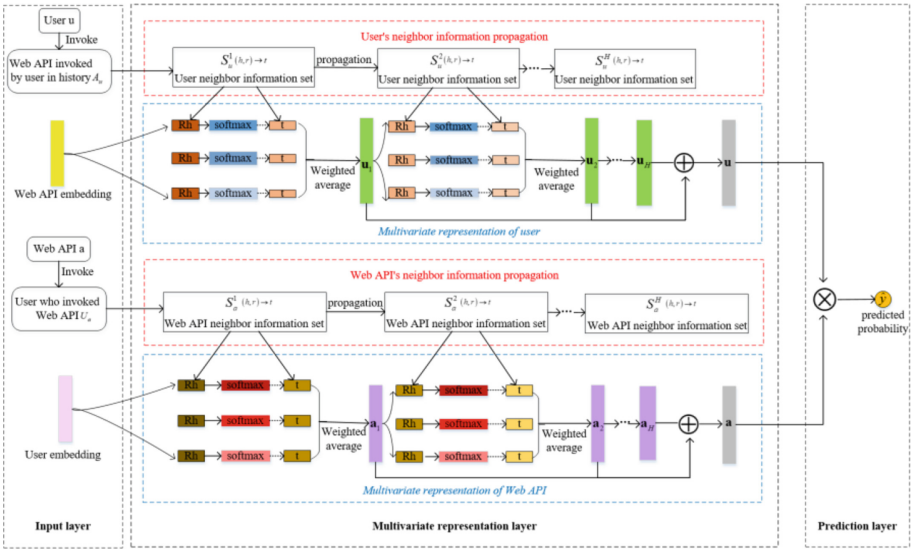


Fig. 1. Overview of the proposed KGWARec.

(1) *Preprocessing layer.* The goal of this layer is to generate an initial embedding vector for a given user and a given Web API. The preprocessing layer makes sufficient preparation for the following multivariate representation layer.
(2) *Multivariate representation layer.* This layer is mainly applied to iteratively calculate the feature representation of users and Web APIs after each layer of propagation through the neighbor information obtained at each hop, so as to propagate and expand the initial neighbor information.
(3) *Prediction layer.* This layer uses the multivariate representations of users and Web APIs learned in the previous layer to predict the probability of users using candidate Web APIs.

4.2 Modeling

Preprocessing Layer

The preprocessing layer mainly uses the user-Web API interaction records to make initial preparation for propagation and calculation of the multi-representation layer. For the input user and Web API, they are initialized by the dense embedding technique. These embedded feature vectors are later sent to

the multivariate representation layer for iterative interaction. Combining all Web APIs used by the user before into a collection, we define the user history usage collection, which can be expressed as: $A_u = \{a \mid Web\ API \quad used \quad by \quad user\, u\}$. The characteristics of Web APIs used by users in history contain the needs of users. Using these Web APIs as initial neighbor nodes can expand and propagate users' interests and explore users' deep intentions.

Similarly, all users who have used the candidate Web API constitute the historical user set of the Web API, $U_a = \{u \mid user \quad who \quad used \quad Web\ API\}$. On the knowledge graph of Web APIs, the neighbors of one Web API can describe the characteristics of the Web API from a certain point of view, and the propagation of neighbor information also starts from the user entity node. The initial neighbor node is also the head entity in the one hop neighbor information set.

Multivariate Representation Layer

Neighbor Information Propagation Based on Knowledge Graph. In the Web API knowledge graph, various types of neighbor entities of users or Web APIs can be found through different kinds of relationships. These relationships and entities contain rich information, which can be used to describe users' interests and the characteristics of Web APIs. We can get the neighbor entities and neighbor information set of each hop by continuously propagating along the relation edges on the knowledge graph. The elements in the neighbor information set are in the form of triples, which contain the relation information between entities. Neighbor entities can be regarded as the natural extension of users' historical interests and Web APIs' characteristics, and contain personalized information of users and Web APIs. Therefore, through the propagation of neighbor information, we can mine users' potential preferences and interests, and capture the semantic association between Web APIs.

In order to explore users' needs through the propagation of neighbor information in the Web API knowledge graph, we recursively define the set of k-hop neighbor entities and k-hop neighbor information set for user u as follows:

Definition 1: *Given user-Web API interaction matrix Y and Web API knowledge graph G, the set of k-hop neighbor entities for user u is defined as:*

$$\mathcal{E}_u^k = \left\{t \middle| (h,r,t) \in \mathcal{G} \quad and \quad h \in \mathcal{E}_u^{k-1}\right\} \tag{2}$$

where $k = 1, 2, \ldots H$, $\epsilon_u^0 = A_u\{y_{ua} = 1\}$ is the set of user u's used Web APIs in the past, which can be seen as the initial neighbor set of user u in Web API knowledge graph G.

Definition 2: *The k-hop neighbor information set of user u is defined as the set of knowledge triples starting from ϵ_u^{k-1}:*

$$S_u^k = \left\{(h,r,t) \,|\, (h,r,t) \in \mathcal{G}, \quad h \in \mathcal{E}_u^{k-1}\right\} \tag{3}$$

where $k = 1, 2, \ldots H$.

Just as exploring the deep needs of users through the propagation of neighbor information in the knowledge graph, Web APIs can also find their corresponding neighbors through the connection with other entities in the knowledge graph, and use neighbor information to accurately describe the characteristics of them. The initial neighbors of one Web API are the users who have used the Web API before. Similarly, we give the definitions of the set of k-hop neighbor entities and k-hop neighbor information set of the Web API:

Definition 3: *Given user-Web API interaction matrix Y and Web API knowledge graph G, the set of k-hop neighbor entities for Web API a is defined as:*

$$\mathcal{E}_a^k = \left\{t | \, (h, r, t) \in \mathcal{G} \quad and \quad h \in \mathcal{E}_a^{k-1}\right\} \tag{4}$$

where k = 1, 2, ...H, $\epsilon_u^0 = A_u\{y_{ua} = 1\}$ *is the set of user's used Web APIs in the past, which can be seen as the initial neighbor set of user u in Web API knowledge graph.*

Definition 4: *The k-hop neighbor information set of Web API a is defined as the set of knowledge triples starting from* ϵ_a^{k-1}*:*

$$S_a^k = \left\{(h, r, t) \, | \, (h, r, t) \in \mathcal{G}, h \in \mathcal{E}_a^{k-1}\right\} \tag{5}$$

where k = 1, 2, ...H.

The size of the neighbor information set refers to the number of neighbor information in the neighbor information set. As the number of hops k increases, the size of the neighbor information set may become too large, which may bring some noisy data, and the influence of the noisy data is greater than that of the valid data. Because the distance between entities after multiple hops is too far from the initial node, the similarity between them decreases, and their features play a relatively small role in users' potential interest exploration and Web APIs' characterization. Therefore, it's necessary to select an appropriate number of hops first, while fixing the size of the neighbor information set to avoid errors and reduce the amount of computation. The selection of specific hop number and neighbor information set size (neighbor size for short) will be discussed in the experiment section.

User Multivariate Representation. The following is a detailed discussion on how to mine users' deep potential interests through knowledge graph neighbor information propagation and obtain users' multivariate representation. As shown in Fig. 1, each Web API a is associated with a d-dimensional embedding $\mathbf{q} \in \mathbf{R}^d$. Given the Web API embedding $\mathbf{q}$ and the one-hop neighbor information set S_u^1 of user u, for each triple (h_i, r_i, t_i) in S_u^1, we compare head h_i and relation r_i with Web API a, and the correlation probability w_i is calculated as follows:

$$w_i = \text{softmax}\left(\mathbf{q}^{\mathrm{T}}\mathbf{R}_i\mathbf{h}_i\right) = \frac{\exp\left(\mathbf{q}^{\mathrm{T}}\mathbf{R}_i\mathbf{h}_i\right)}{\sum_{(h,r,t)\in S_u^1}\exp\left(\mathbf{q}^{\mathrm{T}}\mathbf{R}\mathbf{h}\right)} \tag{6}$$

where $\mathbf{R}_i \in \mathbf{R}^{d\times d}$ is the embedding of relation r_i, $\mathbf{h}_i \in \mathbf{R}^d$ is the embedding of head h_i, w_i represents the similarity between the neighbor entity $\mathbf{h}_i$ and Web API $\mathbf{a}$ measured in the space of relation $\mathbf{R}_i$. Because different relationships will also lead to different similarity between neighbor entity h_i and Web API $\mathbf{q}$, the relationship embedding matrix $\mathbf{R}_i$ should be considered in calculation. Such as Twitter API and Facebook API, if measured by their primary category relation, they are more similar, but when measured by their secondary categories or tags, they are less similar. For all triples in S_u^1, user's potential interest propagates from the head entity h_i to the tail entity t_i along the relation r_i, and take the calculated w_i as the weight to weighted sum of all tail entities. Then the one-order multivariate representation of user u after one-hop propagation is obtained as follows:

$$\mathbf{u}_1 = \sum_{(h_i r_i t_i)\in S_u^1} w_i \mathbf{t}_i \tag{7}$$

where $\mathbf{t}_i \in \mathbf{R}^d$ is the d-dimensional vector embedding of t_i. The user u is represented by his neighbor information in the knowledge graph, rather than an independent feature vector.

Then, the neighbor information of user u continues to propagate. When calculating the two-hop user multivariate representation, we need to replace the $\mathbf{q}$ in (6) with $\mathbf{u_1}$ calculated by (7) and recalculate w_i. For further propagation, $S_u^i, i = 1, 2, ..., H$ for each hop, we repeat the above iterative operation to calculate $\mathbf{u_1}$, $\mathbf{u_2}$, ... $\mathbf{u_H}$. Combined with the user multivariate representation obtained by each hop propagation, the final multivariate representation of user u is obtained as:

$$\mathbf{u} = \mathbf{u}_1 + \mathbf{u}_2 + ... + \mathbf{u}_H. \tag{8}$$

Web API Multivariate Representation. Similar to the method of obtaining multivariate representation, for the multivariate representation of Web APIs, we first obtain the d-dimensional embedded representation vector of each user, and then define the user embeddings and the one-hop neighbor information set S_a^1 with users who have used Web API a as head nodes. For each triple (h'_i, r'_i, t'_i) in S_a^1, we compare head h'_i and relation r'_i with user u to calculate the correlation probability w'_i, which is the similarity between each neighbor node h'_i and user u under relation R'_i:

$$w'_i = \text{softmax}\left(\mathbf{p}^{\mathrm{T}}\mathbf{R}'_i\mathbf{h}'_i\right) = \frac{\exp\left(\mathbf{p}^{\mathrm{T}}\mathbf{R}'_i\mathbf{h}'_i\right)}{\sum_{(\mathrm{h}',r',t')\in S_a^1}\exp\left(\mathbf{p}^{\mathrm{T}}\mathbf{R}'\mathbf{h}'\right)} \tag{9}$$

where $\mathbf{p}$ is the embedding of user u, $\mathbf{R}'_i \in \mathbf{R}^{d\times d}$ is the embedding of relation r'_i, $\mathbf{h}'_i \in \mathbf{R}^d$ is the embedding of head h'_i.

Web APIs obtain various types of neighbors through the triple propagation in KG. The information of neighbors arriving at each layer is aggregated to enrich the feature representation of Web APIs. After one-hop propagation, the multivariate representation of Web API a is:

$$\mathbf{a_1} = \sum_{(h'_i r'_i t_i') \in S_a^1} w'_i \mathbf{t}'_i \tag{10}$$

where $\mathbf{t}'_i \in \mathbf{R}^d$ is the d-dimensional vector expression.

Continuing to propagate along the relation edges in the Web API knowledge graph to obtain the neighbor characteristics of the next layer of Web API a, we use $\mathbf{a_1}$ to replace $\mathbf{u}$ in (9), recalculate according to the above steps and repeat the iterative process to obtain Web API a's multivariate representation $\mathbf{a_1}$, $\mathbf{a_2}$, ... $\mathbf{a_H}$ of each layer through H-hops. Finally, by overlaying the results of each layer, we get Web API a's multivariate representation obtained through the propagation of the knowledge graph's neighbor information:

$$\mathbf{a} = \mathbf{a}_1 + \mathbf{a}_2 + ... + \mathbf{a}_H \tag{11}$$

Prediction Layer

According to multivariate representation layer, we get the final multivariate representation $\mathbf{u}$ and $\mathbf{a}$ of user u and Web API a. The user multivariate representation $\mathbf{u}$ contains the history information of user u and its neighbor information in the knowledge graph. The Web API multivariate representation $\mathbf{a}$ contains the API itself and its neighbor information in the knowledge graph. Then we use the inner product of user embedding and Web API embedding to get the predicted probability value:

$$\hat{y}_{ua} = \sigma\left(\mathbf{u}^T \mathbf{a}\right) \tag{12}$$

The loss function for KGWARec is as follows:

$$\begin{aligned} \min \mathcal{L} = \sum_{(u,a) \in \mathbf{Y}} \mathcal{T}\left(\hat{y}_{ua}, y_{ua}\right) + \frac{\lambda_2}{2} \sum_{r \in \mathcal{R}} \left\| \mathbf{I}_r - \mathbf{E}^{\mathrm{T}} \mathbf{R} \mathbf{E} \right\|_2^2 + \\ \frac{\lambda_1}{2} \left(\|\mathbf{A}\|_2^2 + \|\mathbf{U}\|_2^2 + \|\mathbf{E}\|_2^2 + \sum_{r \in \mathcal{R}} \|\mathbf{R}\|_2^2 \right) \end{aligned} \tag{13}$$

where $\mathbf{A}$ and $\mathbf{E}$ are the embedding matrices for all Web APIs and entities respectively, and $\mathbf{I}_r$ is the slice of the indicator tensor $\mathbf{I}$ in KG for relation r. In (13), the first term is the cross entropy-loss function, which measures the difference between the real value of the interaction matrix $\mathbf{Y}$ and the predicted value by KGWARec, the second term is the loss for knowledge graph feature representation, and the third term is the parameter regularization loss, which is to prevent over-fitting.

5 Experiments

5.1 Preparation

Datasets. We crawl Web API-related data from the ProgrammableWeb platform, and use historical interaction records between users and Web APIs as the experimental dataset. The statistics of the experimental dataset are shown in Table 2. In addition, under the premise of ensuring that the historical needs of users are not changed, negative sampling is performed to reduce the imbalance of positive and negative samples. This specific method is to randomly select a set of Web APIs with no user behavior from all Web APIs, whose amount is equal to the amount of data marked as 1, and mark the records composed of APIs from the set and the users as 0 as negative samples.

All experiments are conducted using python-3.6.5 in Windows 10 OS. The configuration is CPU i7-7700 @ 3.60 GHz, 8G RAM. The ratio of training set, evaluation set, and test set is 6:2:2.

Table 2. Statistics for experimental dataset

Item	Count
Web API	1,163
User	2,701
Interaction	15,184
Sparsity	0.9953

5.2 Comparison

The following methods are selected as the baselines:

FM [25], considers the interaction between features and models the interaction of all nested variables. It can estimate reliable parameters in the case of sparse data, so as to obtain the interaction prediction probability of user items.

AFM [26], introduces the attention mechanism into the feature crossover module to learn the importance of different crossover features. The attention score calculated by the attention network is used to weighted sum the interactive features, and the FM model is extended.

NFM [27], is a neural network attempt of FM model. It combines the linear intersection of FM to second-order features with the nonlinear intersection of neural network to higher-order features to strengthen the expression ability of the model.

MKR [28], automatically learns the high-order feature interaction between the items in the recommendation system and the entities in the knowledge graph through the cross-compression unit, and uses the multi task learning framework for alternating learning to improve the recommendation quality.

KGCN [29], combines the features of knowledge graph with graph convolution neural network, enriches the embedded representation of items by aggregating neighbor information, catch local neighborhood structure, and obtains the personalized preferences of user.

KGNN-LS [30], develops a method based on label smoothness, designs a loss function for label propagation, regularizes the edge weight in the process of learning, and realizes better generalization.

The compared results are presented in Table 3.

Table 3. AUC and ACC results of different models

Model	AUC	ACC
FM	0.8460	0.7621
AFM	0.8438	0.7663
NFM	0.8467	0.7370
MKR	0.7849	0.7906
KGCN	0.8502	0.8143
KGNN-LS	0.8779	0.8277
KGWARec	0.9372	0.8763

(1) It can be seen from Table 3, our KGWARec model achieves the best results on the evaluation metrics AUC and ACC. Compared with the KGNN-LS model with the best performance in the baseline model, the AUC and ACC improvements of KGWARec is 5.7% and 4.9% respectively, indicating the utilization of structural information and semantic information of knowledge graph can achieve more accurate predictions under sparse scenarios.

(2) By observing the results of ACC in the table, it can be seen that the performance of MKR, KGCN, KGNN-LS and KGWARec, that is, models integrated with the knowledge graph, perform better than the FM, AFM and NFM without knowledge graph. This indicates that leveraging knowledge graph as auxiliary information to recommendation can indeed provide richer auxiliary information for recommendation tasks, optimize feature representation, and improve recommendation accuracy.

(3) FM, AFM and NFM models also achieve better results in the case of sparse interaction data, but the performance is not as good as the proposed KGWARec model. The reason is that although models such as FM take into account the intersection between features, but these models ignore the semantic information of the relationship between features, while KGWARec focuses on the role of different relational spaces.

(4) KGCN, KGNN-LS and KGWARec have better performance than MKR, which can show that mining knowledge graph by capturing neighbor information in knowledge graph is effective. In addition, KGWARec utilizes the

method of neighbor information propagation on the knowledge graph, and enriches the multiple representations of users and Web APIs, which further improves the recommendation performance.

(5) Figure 2 shows the performance of each model under the evaluation Recall@K. K is the number of selected Web APIs with the highest predicted value, and the value of K is 1, 10, 20, 30, 40, 50. It can be seen that KGWARec also shows the best performance under the Recall@K indicator.

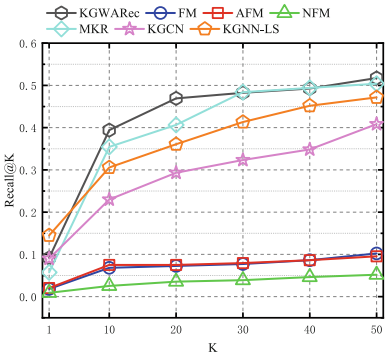


Fig. 2. The results of Recall@K.

5.3 Ablation Study

In this experiment, we study the impact of knowledge graph on Web API recommendation. KGWARec-None indicates not using knowledge graph, KGWARec-U and KGWARec-A indicate multi-representation of user and Web API using knowledge graph, respectively. Experimental results are shown in Table 4.

Table 4. AUC and ACC results with different ablation strategy

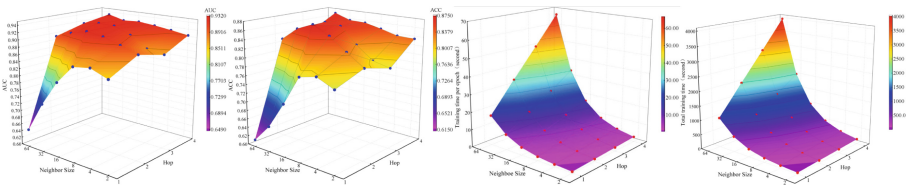
Model	AUC	ACC
KGWARec-None	0.7121	0.4529
KGWARec-U	0.8166	0.6580
KGWARec-A	0.8701	0.7597
KGWARec	0.9372	0.8763

It can be seen from Table 4 that the performance of KGWARec-None is the worst, ACC has not reached 0.5, which means that the classifier is unqualified.

The reason is that user Web API interaction data is very sparse, KGWARec-None do not use any auxiliary information of the knowledge graph, it is difficult to characterize user's interest or Web API's feature through the sparse interaction. Compared with KGWARec-None, the AUC and ACC results obtained by KGWARec-U and KGWARec-A are significantly higher, which well proves that the neighbor information propagation method introduced into the knowledge graph can effectively improve the data sparsity problem. KGWARec obtains the best results, compared with KGWARec-A, the AUC is increased by 7.7%, and the ACC is increased by 15.3%, which further proves the theoretical validity of this model. The addition of structural information and semantic information will have a positive impact on the recommendation, it is very meaningful to enrich the multiple representations of users and Web APIs at the same time.

5.4 Impact of Hop Number and Neighbor Size

The hop number and neighbor size are the two main factors of our model. Specifically, if the number of hops and neighbor size is too small, the correlation and dependency between entities cannot be explored, and if the hop number and and neighbor size is too large, it will bring some noise data and affect the performance of the algorithm. In order to get the appropriate hop and neighbor size, we change the hop and neighbor size at the same time to observe the changes of evaluation indicators AUC and ACC, which not only ensures the best representation of the characteristics of Web API and users' interests, but also avoids the influence of too much noise information.



(a) AUC (b) ACC (c) Time per epoch (d) Training Time

Fig. 3. Impact of hop number and neighbor size

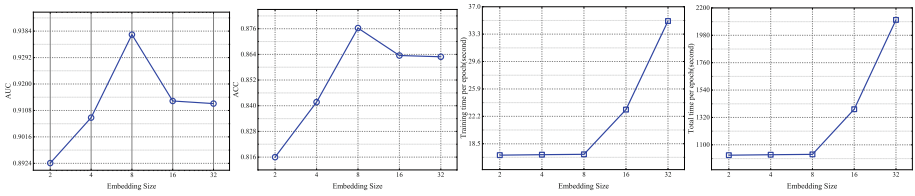
By observing Fig. 3(a), it can be seen that when the number of hops is selected as 1, the value of AUC is low, and then it first increases and then decreases with the increase of the number of hops. The change trend of AUC with the size of neighbors also increases first and then decreases. At the same time, it can be observed that the AUC results are better when the hop count is 2, the neighbor size is 16, the hop count is 3, the neighbor size is 32, and the hop count is 4, and the neighbor size is 64. It can be clearly observed in Fig. 3(b) that the ACC reaches a maximum of 0.8749 when the hop count is set to 2 and the neighbor size is set to 16. Therefore, although the AUC values obtained at 3

hops of 32 neighbors and 4 hops of 64 neighbors increase slightly for 2 hops of 16 neighbors, the magnitude is very small, while the ACC is at 3 hops of 32 neighbors and 4 hops of 64 neighbors. There is a significant decrease in neighbors, so the performance of AUC and ACC is comprehensive, and it is believed that setting the number of hops to 2 and the size of neighbors to 16 can make the model reach the optimum.

Figure 3(c) and Fig. 3(d) shows the effect of the hop and neighbor size on training time per epoch and the total training time. It can be obviously observed that the training time is increased with the rising of hops and neighbor size, which demonstrates that the efficiency of KGWARec is affected by hop and neighbor size.

5.5 Impact of Embedding Size

In this section, we study the effect of embedding size on the performance of our model KGWARec. We adjusted the embedding size from 2 to 32, while keeping the other parameters unchanged. The values of AUC and ACC are shown in Fig. 4(a) and Fig. 4(b).



(a) AUC (b) ACC (c) Time per epoch (d) Training Time

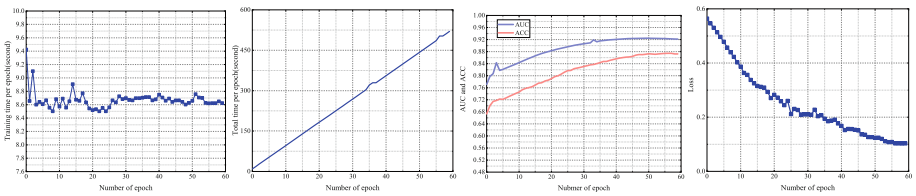
Fig. 4. Impact of hop embedding size

It can be observed from the research results that the accuracy of our KGWARec model has been improved at the beginning, because more useful information can be obtained by appropriately increasing the embedded dimension. When the embedding size approach 8, the best experimental results are reached, and then it begins to decline. Figure 4(c) and Fig. 4(d) show the values of training time per epoch and total training time under different embedding sizes. We can clearly see that training time is linear with the increasing of embedding size, illustrating the efficient of our KGWARec model related to embedding size.

5.6 Convergence of KGWARec

To evaluate the efficiency and convergence of KGWARec, we observe the training time of KGWARec and investigate whether the prediction results converge

through the increasing of training epoch. Figure 5(a) and (b) report the training time of each epoch and the total training time of KGWARec respectively. Figure 5(c) and (d) present the values of AUC and ACC for each epoch, and the training loss as the epoch grows, respectively.



(a) Time per epoch (b) Training time (c) AUC and ACC (d) Loss

Fig. 5. Convergence of KGWARec

As can be seen in Fig. 5(a), the training time of each epoch fluctuates in a small range, which show that our algorithm is stable. In Fig. 5(b), the total training time increases linearly with the training epoch, showing that the training epoch is also one of the factors affecting the efficiency of our algorithm. By observing the curve in Fig. 5(c), it can be see that, the value of AUC and ACC first increase significantly and then gradually stabilized with the increase of epoch. The loss curve in Fig. 5(d) first decreases rapidly and then tends to be stable with the increase of epoch. These results can strongly prove that our method has good convergence within the effective training times.

6 Conclusion

In this paper, we presented a knowledge graph-enhanced Web API recommendation model based on neighborhood information propagation method. To this end, we first use the crawled real-world Web API data to build a Web API knowledge graph, and then propose a neighbor information propagation method to make full use of the higher-order structural information and contextual semantic information in the knowledge graph for obtaining the multivariate representation of users and Web APIs. The probability of users' interest to the target Web API is predicted through the multivariate representation of users and Web API. KGWARec uses an end-to-end joint training method to update parameters, no manual feature extraction is needed. Finally, experimental results demonstrate the superiority and effectiveness of our KGWARec model. And it proves that improving Web API recommendation performance by incorporating with knowledge graph neighbor information is a correct research route. In the near future, we plan to further improve the accuracy of Web API recommendation from the perspective of API complementarity by using knowledge graph.

References

1. Zhang, L., Zhang, J., Cai, H.: Services Computing. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-38284-3
2. Niknejad, N., Ismail, W., Ghani, I., Nazari, B., Bahari, M., et al.: Understanding Service-Oriented Architecture (SOA): a systematic literature review and directions for further investigation. Inf. Syst. **91**, 101491 (2022)
3. Hustad, E., Olsen, D.: Service-oriented architecture. Creating a sustainable digital infrastructure: the role of service-oriented architecture. Procedia Comput. Sci. **181**, 597–604 (2021)
4. Tang, B., Yan, M., Zhang, N., et al.: Co-attentive representation learning for web services classification. Expert Syst. Appl. **180**, 115070 (2021)
5. Qi, L., Song, H., Zhang, X., et al.: Compatibility-aware web API recommendation for mashup creation via textual description mining. ACM Trans. Multimedia Comput. Commun. Appl. **17**(1s), 1–19 (2021)
6. Adeleye, O., Yu, J., Wang, G., et al.: Constructing and evaluating evolving web-API networks-a complex network perspective. IEEE Trans. Serv. Comput. (2021)
7. Ebesu, T., Shen, B., Fang, Y.: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 515–524 (2018)
8. Cui, Z., Xu, X., Fei, X., Cai, X., et al.: Personalized recommendation system based on collaborative filtering for IoT scenarios. IEEE Trans. Serv. Comput. **13**(4), 685–695 (2020)
9. Yi, B., Shen, X., Liu, H., et al.: Deep matrix factorization with implicit feedback embedding for recommendation system. IEEE Trans. Industr. Inf. **15**(8), 4591–4601 (2019)
10. Chen, C., Zhang, M., Ma, W., et al.: Efficient non-sampling factorization machines for optimal context-aware recommendation. In: Proceedings of the Web Conference 2020, pp. 2400–2410 (2020)
11. Tang, M., Jiang, Y., Liu, J., et al.: Location-aware collaborative filtering for QoS-based service recommendation. In: 2012 IEEE 19th International Conference on Web Services, pp. 202–209(2012)
12. Chen, K., Mao, H., Shi, X., et al.: Trust-aware and location-based collaborative filtering for Web service QoS prediction. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 143–148 (2017)
13. Zhang, Y., Wang, K., He, Q., et al.: Covering-based web service quality prediction via neighborhood-aware matrix factorization. IEEE Trans. Serv. Comput. **14**(5), 1333–1344 (2019)
14. Fletcher, K.K.: A quality-aware web API recommender system for mashup development. In: Ferreira, J.E., Musaev, A., Zhang, L.-J. (eds.) SCC 2019. LNCS, vol. 11515, pp. 1–15. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23554-3_1
15. Chen, Z., Shen, L., Li, F.: Your neighbors are misunderstood: on modeling accurate similarity driven by data range to collaborative web service QoS prediction. Futur. Gener. Comput. Syst. **95**, 404–419 (2019)
16. Jannach, D., Lerche, L., Zanker, M.: Recommending based on implicit feedback. In: Brusilovsky, P., He, D. (eds.) Social Information Access. LNCS, vol. 10100, pp. 510–569. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90092-6_14
17. Xiong, R., Wang, J., Zhang, N., et al.: Deep hybrid collaborative filtering for web service recommendation. Expert Syst. Appl. **110**, 191–205 (2018)

18. Cao, Y., Liu, J., Shi, M., et al.: Service recommendation based on attentional factorization machine. In: 2019 IEEE International Conference on Services Computing (SCC), pp. 189–196 (2019)
19. Zhao, H., Wang, J., Zhou, Q., Wang, X., Wu, H.: Web API recommendation with features ensemble and learning-to-rank. In: Jin, H., Lin, X., Cheng, X., Shi, X., Xiao, N., Huang, Y. (eds.) BigData 2019. CCIS, vol. 1120, pp. 406–419. Springer, Singapore (2019). https://doi.org/10.1007/978-981-15-1899-7_29
20. Huang, J., Zhao, W.X., Dou, H., et al.: Improving sequential recommendation with knowledge-enhanced memory networks. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 505–514 (2018)
21. Kwapong, B., Fletcher, K.: A knowledge graph based framework for web API recommendation. In: 2019 IEEE World Congress on Services (SERVICES), vol. 2642, pp. 115–120 (2019)
22. Kwapong, B., Anarfi, R., Fletcher, K.K.: A knowledge graph approach to mashup tag recommendation. In: 2020 IEEE International Conference on Services Computing (SCC), pp. 92–99 (2020)
23. Wang, X., Wu, H., Hsu, C.H.: Mashup-oriented API recommendation via random walk on knowledge graph. IEEE Access **7**, 7651–7662 (2018)
24. Geng, J., Cao, B., Ye, H., et al.: Web service recommendation based on knowledge graph convolutional network and Doc2Vec. In: 2020 IEEE World Congress on Services (SERVICES), pp. 95–100 (2020)
25. Rendle, S.: Factorization machines. In: 2010 IEEE International Conference on Data Mining, pp. 995–1000 (2010)
26. Xiao, J., Ye, H., He, X., et al.: Attentional factorization machines: learning the weight of feature interactions via attention networks. arXiv preprint arXiv:1708.04617 (2017)
27. He, X., Chua, T.S.: Neural factorization machines for sparse predictive analytics. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 355–364 (2017)
28. Wang, H., Zhang, F., Zhao, M., et al.: Multi-task feature learning for knowledge graph enhanced recommendation. In: The World Wide Web Conference, pp. 2000–2010 (2019)
29. Wang, H., Zhao, M., Xie, X., et al.: Knowledge graph convolutional networks for recommender systems. In: The World Wide Web Conference, pp. 3307–3313 (2019)
30. Wang, H., Zhang, F., Zhang, M., et al.: Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 968–977 (2019)